Stata für Umsteiger: ein Einführungskurs

Kohler, Ulrich

Historical Social Research
Historische Sozialforschung

HSR Trans 11

Ulrich Kohler

Stata für Umsteiger. Ein Einführungskurs

doi: 10.12759/hsr.trans.11.v01.2002

Version: 22 October 2002
HSR Trans 11 (2002)

Stata für Umsteiger

Ulrich Kohler
Universität Mannheim

22. Oktober 2002

1 Wie man diesen Text benutzt

Dieser Kurs soll das Umsteigen von anderen Datenanalyseprogrammen zu Stata erleichtern. Aus diesem Grund werden die besonderen Eigenschaften von Stata hervorgehoben und Eigenschaften, die Stata mit anderen Datenanalyseprogrammen gemeinsam hat, nicht oder nur am am Rande behandelt.

Im Text werden nur sehr wenig Befehle besprochen. Der Schwerpunkt der Darstellung liegt auf den allgemeinen Strukturen von Stata. Diese können besser gesehen werden, wenn einige wenige Befehle mit den Möglichkeiten der Syntax immer wieder verändert werden. Um die Befehle für spezielle statistische Verfahren herauszufinden kann man sich der Suchfunktion innerhalb von Stata bedienen. Diese ist über das Hilfe–Menü oder den Befehl `search` aufrufbar.

Der Aufbau des Textes ist wie folgt: Jede Seite beginnt mit einer einführenden *Folie*. Darauf werden die wesentlichen Punkte zu einem bestimmten Thema dargestellt und konkrete Beispiele vorgeführt. Einige der Folien sind selbsterklärend. Die meisten Folien werden jedoch durch einen kurzen Text erläutert. Es ist beabsichtigt, dass die Folien für sich genommen nach dem ersten Durcharbeiten des Kurses eine Kurzzusammenfassung darstellen.

Der Text enthält *keine* Outputs. Die Ergebnisse der Beispiele werden also nicht im Text vorgeführt. Für das Verständnis ist es *unbedingt* erforderlich, dass alle Beispiele am Rechner nachvollzogen werden. Alle aufgeführten Befehle sollten *eingegeben* und das Resultat sorgfältig *betrachtet* werden. Man sollte erst dann zum nächsten Schritt weitergehen, wenn man die vorangegangenen Befehle *verstanden* hat. Um diese aktive Mitarbeit zu ermöglichen wurde für den Kurs ein Datensatz verwendet, der zusammen mit Stata ausgeliefert wird. Sie finden ihn normalerweise im Stata–Programmverzeichnis.

Zum Schluß noch ein Wort zum Zeitaufwand. Der gesamte Kurs sollte in etwa 4 Stunden durchgeführt werden können. Der Kurs dauert länger wenn man — was sinnvoll ist — außer den Beispielen im Text weitere Befehle bzw. Variationsmöglichkeiten ausprobiert. In jedem Fall kann man den Kurs an jeder Stelle unterbrechen. Hierzu verläßt man Stata ohne zu speichern. Nach dem Aufrufen des Datensatzes wird der Kurs an der entsprechenden Stelle einfach fortgesetzt.

2 Allgemeine Grundlagen

Slide 1

Stata aufrufen und Stata verlassen

Der Aufruf von Stata erfolgt durch:

1. Klicken auf **Start**.
2. Klicken auf **Programme**.
3. Klicken auf **Stata**.
4. Wählen von **Intercooled Stata, Pseudo-Intercooled Stata** oder **Small Stata**.

Das Verlassen von Stata erfolgt durch

1. Klicken auf **File**.
2. Klicken auf **Exit**.

oder durch Eingabe des Wortes `exit` in das Kommandofenster.

Dies gilt für die Betriebssysteme Windows 98/95/NT. Unter anderen Betriebssystemen gelten die Konventionen des jeweiligen Betriebssystems.

Beim Aufruf von Stata können beliebige Befehle automatisch ausgeführt werden und so die Eigenschaften von Stata den individuellen Bedürfnissen angepaßt werden. Dazu werden die entsprechenden Befehle werden in die Datei *profile.do* geschrieben. Die Datei *profile.do* wird im Rahmen der *Do–Files* ausführlich besprochen.

Die Eingabe von Befehlen

Stata wird durch eine Kommandosprache gesteuert. In der Kommandozeile werden Befehle als Worte, Buchstaben oder Zahlen eingegeben.

▷ **Beispiel**

Slide 2

Schreiben Sie das Wort `describe` in das Kommandofenster. Nach drücken der Eingabetaste erscheint folgende Ausgabe:

```
. describe

Contains data
  obs:             0
 vars:             0
 size:             0 (99.3% of memory free)
Sorted by:
```

Das Wort `describe` ist ein Befehl. Mit dem Befehl `describe` kann man Datensätze näher beschreiben. Da im Augenblick noch kein Datensatz geladen ist, ist das uninteressant. Interessant ist nur, dass in Stata durch ein oder mehrere Buchstaben bzw. Worte Befehle eingegeben werden. Die Eingabe eines Befehls wird mit der Eingabetaste abschlossen. Danach wird der eingegebene Befehl im *Ergebnisfenster* wiederholt und es erfolgt die Ausgabe des Befehls.

▶ Immer wenn Sie nachfolgend ein Wort in `dieser Schrift` sehen, dass durch einen „Punkt“ eingeleitet wird, sollten Sie das Wort in das Kommandofenster oder hinter die Eingabeaufforderung schreiben und danach die Eingabetaste drücken. Tippen Sie das Wort ohne den einleitenden Punkt ein und übernehmen Sie die Groß- und Kleinschreibung. Stata ist *„case–sensitve“* d.h. Großbuchstaben haben eine andere Bedeutung als Kleinbuchstaben.

Slide 3

Daten laden

Mit dem Befehl `use` werden Stata–Systemdatensätze geladen. Nach dem Kommando wird der Dateiname angegeben. Wenn keine Erweiterung angegeben wird, wird „dta“ vorausgesetzt.

Ansonsten gelten die Konventionen des jeweiligen Betriebssystems.

▷ **Beispiel**

```
. use auto
. use c:/programme/stata/auto
. use "c:/user/daten/stata/Automobile 1978"
. use http://www.stata.com/manual/oddeven.dta
```

Bevor Sie beginnen: Sie brauchen die Befehle nicht jedesmal komplett einzutippen. Mit der *„Bild–nach–oben“* Taste können Sie die zuvor eingegebenen Befehle wieder in das Kommandofenster zurückholen. Unter Windows können Sie dasselbe auch mit einem Doppelklick auf die Befehle im *Review–Fenster* erreichen.

Die gezeigten Befehle führen in den meisten Fällen zur Fehlermeldung

```
file xyz.dta not found
r(601);
```

Mit dem ersten Befehl wird der Datensatz „auto.dta“ geladen. Der Datensatz wird im aktuellen Arbeitsverzeichnis gesucht. (Wahrscheinlich ist im aktuellen Arbeitsverzeichnis keine Datei mit dem Namen *auto.dta*. Deshalb wird eine entsprechende Fehlermelung angezeigt.)

Mit dem zweiten Befehl wird der Datensatz „auto.dta“ aus dem Verzeichnis c:/programme/stata geladen. Unter Windows kann statt des vorwärtsgewandten Schrägstrichs auch der *backslash* verwendet werden. (Wenn Stata vollständig in das Verzeichnis c:/programme/stata installiert wurde, wird dieser Befehl ausgeführt. Ansonsten erscheint eine entsprechende Fehlermeldung.)

Im dritten Befehl wird ein *langer* Variablennamen verwendet. In diesem Fall muß der Variablennamen in Anführungszeichen angegeben werden. (Wahrscheinlich ist im angegebenen Verzeichnis keine Datei mit dem Namen *automobile 1978.dta*. Deshalb wird eine entsprechende Fehlermelung angezeigt.)

Im vierten Befehl wird ein Datensatz direkt aus dem Internet geladen. Dazu wird einfach die Adresse als Dateinamen eingegeben. (Dieser Befehl funktioniert nur, wenn Sie *online* sind)

▶ Der nachfolgende Text nimmt Bezug auf den Datensatz „auto.dta“. Dieser wird mit Stata ausgeliefert und befindet sich normalerweise im Stata–Programmverzeichnis z.B. in `c:/programme/stata`. Bitte laden Sie diesen Datensatz. Nach dem Laden können Sie sich mit

```
. describe
```

einen Überblick über die Daten verschaffen. Es handelt sich um einige Angaben zu 74 Autos aus dem Jahr 1978. Darunter z.B. der Preis, der Verbrauch in *Meilen pro Gallone*, der Hubraum usw.

Slide 4

Syntax der Befehle

Fast alle Stata–Kommandos bestehen aus folgenden, in spezifischer Reihenfolge angeordneten Bausteinen:

```
[Prefix :] Befehl [Variablenliste] [if] [in] [Gewicht] [, Option]
```

Durch die Online–Hilfe bekommt man einen Hilfetext zu den Bausteinen:

```
. h varlist
. h if
. h in
. h weights
. h options
```

Den Kern bildet der eigentliche *Befehl*. Ihm können, abgetrennt durch einen Doppelpunkt, *Befehlsprefixe* vorangestellt werden. Es gibt im wesentlichen zwei solcher Prefixe, `by:` und `for:`. Direkt an den Befehl angeschlossen wird eine *Variablenliste*. Danach folgen in beliebiger Reihenfolge die Angabe über die etwaige Gewichtung des Datensatzes die *„if“–Bedingung* und die *„in“–Bedingung*. Schließlich werden, abgetrennt durch ein Komma, die *Optionen* eingegeben.

▶ Im folgenden werden die Eigenschaften der Bausteine beschrieben. Beachten Sie, dass diese Eigenschaften für jedem Befehl dieselben bleiben.

Slide 5

Welcher Baustein ist erlaubt?

Man kann einen Baustein nur verwenden, wenn er für einen Befehl *zugelassen* oder *vorgeschrieben* ist. Die Information darüber, erhalten Sie über die Syntax–Angabe der Online–Hilfe. Ein Baustein gilt als zugelassen, wenn er bei der Syntax–Angabe genannt wird. Erfolgt die Nennung *nicht* in eckigen Klammern, so ist der Baustein zwingend vorgeschrieben. Bausteine, die nicht angegeben werden, sind nicht zugelassen.

▷ **Beispiel**

Die Syntaxangabe des Befehls `summarize` lautet:

```
[by varlist:] summarize [varlist] [weight] [if exp] [in range]
                                [, {detail | meanonly } format]
```

Alle oben genannten Bausteine sind zugelassen, nur der Befehl ist zwingend vorgeschrieben.

Slide 6

Der Befehl

Mit dem Befehl wird festgelegt, welche Prozedur angefordert wird. Manche Befehle können abgekürzt werden. In der Online–Hilfe wird die maximal mögliche Abkürzung farblich hervorgehoben. Zwischen der maximal möglichen Abkürzung und dem Ausschreiben des gesamten Befehls sind alle Variationen möglich:

▷ **Beispiel**

```
. su
. sum
. summ
. summa
. summar
. summari
. summariz
```

Mit dem Befehl `summarize` werden deskriptive Maßzahlen wie Mittelwert und Standardabweichung angefordert. Einen knappen Überblick über weitere Befehle von Stata kann man sich übrigens durch

```
. help contents
```

verschaffen. Hilfe zu den einzelnen Befehlen erhält man durch Eingabe des Befehls `help` und dem jeweiligen Befehl, z.B.

```
. help summarize
```

`help` kann durch `h` abgekürzt werden.

Übrigens: Wenn Sie nicht durch die ganze Hilfe blättern wollen können Sie die Ausgabe mit *Strg+Pause* abbrechen. Mit *Strg+Pause* kann jeder Befehl unterbrochen werden.

Slide 7

Die Variablenliste

Die Variablenliste ist eine von Leerzeichen unterbrochene Liste von Variablennamen. Anzahl und Reihenfolge der Variablen spielen dabei keine Rolle.

Bei einigen Befehlen *kann* eine Variablenliste angegeben werden, ohne dass dies zwingend vorgeschrieben ist. Bei anderen Befehlen muß eine Variablenliste angegeben werden.

▷ **Beispiel**

```
. summarize mpg weight
. sum
. regress mpg weight
. reg
. tabulate for rep78
. tab
```

Wenn die Angabe einer Variablenliste nicht zwingend vorgeschrieben ist, kann es sein, daß der Befehl im Falle des Weglassens der Variablenliste für alle Variablen durchgeführt wird.

Bei anderen Befehlen führt das Weglassen der Variablenliste zur Wiederholung des Befehls mit der zuletzt beim selben Befehl eingegebenen Variablenliste. Dies gilt insbesondere für *alle* Befehle zur Berechnung von Modellen (z.B. `regress` für lineare Regressionsmodelle).

Bei einigen Befehlen muß eine Variablenliste spezifiziert werden. Dies ist dann der Fall, wenn eine Wiederholung oder Anwendung auf alle Variablen nicht möglich oder nicht sinnvoll ist. Dies ist z.B. der Fall bei `tabulate`, dem Befehl zur Darstellung ein- oder zweidimensionaler Häufigkeitstabellen.

Slide 8

Abkürzungen der Variablenliste

Variablennamen können abgekürzt werden. Es genügt nur so viel des Namens einzugeben, dass die Variable eindeutig identifizierbar wird.

▷ **Beispiel**

```
. sum mp w
. tab f r
```

Nur die Variable *weight* fängt mit dem Buchstaben *w* an. Mit dem Buchstaben *m* fangen dagegen die Variablen *make* und *mpg* an. Hier muß darum mindestens *mp* verwendet werden um *mpg* zu identifizieren.

Slide 9

Angabe von Variablenbereichen

In Variablenlisten gibt es zwei Möglichkeiten Variablenbereiche zu spezifizieren:

- Variablen mit gleichen Anfangsbuchstaben können durch die *Wildcard* „*“ gemeinsam angesprochen werden.
- Variablen, die im Datensatz hintereinander stehen, können durch einen Bindestrich gemeinsam angesprochen werden.

▷ **Beispiel**

```
. sum m*
. sum ma-re
```

Slide 10

Die „in“–Bedingung

Mit der „in“–Bedingung wird die Ausführung eines Befehls auf bestimmte Fälle eingegrenzt. Die „in“–Bedingung nimmt Bezug auf die Position der Fälle im Datensatz.

▷ **Beispiel**

```
. sort mpg
. list make mpg price in 10
. l make mpg price in 10/25
. l make mpg price in -10/-1
```

Mit `sort` wird der Datensatz in aufsteigender Reihenfolge der Werte der angegebenen Variable sortiert. `list` zeigt die Werte der angegebenen Variablen fallweise an.

Das erste `list`–Kommando zeigt die Werte der Variablen *make*, *mpg* und *price* für den zehnten Fall im Datensatz. Das zweite Kommando zeigt die entsprechenden Werte für den 10. bis 25. Fall. Das dritte zeigt den Preis des zehntletzten bis letzten Falls. Da der Datensatz nach dem Verbrauch sortiert ist, handelt es sich hierbei um die Wagen mit dem niedrigsten Kraftstoffverbrauch.

Slide 11

Die „if“–Bedingung

Mit der „if“–Bedingung wird die Ausführung eines Befehls auf diejenigen Fälle eingegrenzt, für die der in der „if“–Bedingung gegebene *Ausdruck* wahr ist.

Ein einfaches Beispiel für eine „if“–Bedingung ist:

```
. sum mpg if foreign==1
```

Hier wird der Befehl `sum mpg` für diejenigen Fälle ausgeführt, für die der Ausdruck „foreign==1“ *wahr* ist.

In „if“–Bedingungen können wesentlich kompliziertere Ausdrücke verwendet werden als in diesem Beispiel. Die Variationsmöglichkeiten ergeben sich durch Verwendung verschiedenster Operatoren und Funktionen. Als Operatoren stehen zur Verfügung:

```
    Arithmetic              Logical            Relational
                                            (numeric and string)
-------------------     -------------------     --------------------
+    addition               ~    not           >    greater than
-    subtraction            |    or            <    less than
*    multiplication         &    and           >=   > or equal
/    division                                  <=   < or equal
^    power                                     ==   equal
                                               ~=   not equal
+    string concatenation

   Note that a double equal sign (==) is used for equality testing.
```

Eine Liste von Funktionen findet man durch Eingabe von

```
. help functions
```

Slide 12

Variationsmöglichkeiten von if–Bedingungen

Durch die Verwendung von Operatoren und Funktionen ergeben sich zahlreiche Variationsmöglichkeiten von if–Bedingungen

▷ **Beispiel**

```
. sum mpg if weight > 700
. sum mpg if weight > 700 & for == 0
. list make if (100*3.78)/(mpg*1.6) >= 10 & for == 1
. sum mpg if uniform() > .5
```

Der erste Befehl zeigt eine einfache Verwendung einer größer/kleiner Bedingung. Im zweiten Befehl wird zusätzlich ein logischer Operator verwendet. Der Ausdruck im dritten Befehl sorgt dafür, dass diejenigen nicht–amerikanischen Autos angezeigt werden, die über 10 *Liter* auf 100 *km* verbrauchen. Im letzten Befehl wird Statas Funktion für den Zufallszahlengenerator verwendet. `uniform()` steht für gleichverteilte Zufallszahlen zwischen 0 und 1. Der letzte Befehl zeigt also den Durchschnitt von ungefähr 37 zufällig ausgewählten Autos.

Slide 13

Eine Warnung

Fehlende Werte werden in Stata auf $+\infty$ gesetzt. Sie werden darum durch Ausdrücke mit den relationalen Operatoren für $>$ und $\geq$ eingeschlossen.

▷ **Beispiel**

```
. tab rep78
. sum mpg if rep78 >= 4
```

Es wird der durchschnittliche Verbrauch von 34 Autos dargestellt, obwohl es laut Häufigkeitstabelle nur 29 Autos mit einem *Repair–Record* von 4 oder darüber gibt. Grund: Für die restlichen 5 Autos ist der *Repair–Record* unbekannt, also $+\infty$.

Wenn der Durchschnitt *ohne* die Fälle mit unbekanntem „Repair–Record“ berechnet werden soll, muß die „if“–Bedingung näher spezifiziert werden. Dies geschieht durch Verknüpfung mehrerer Bedingungen mit den „logischen“ Operatoren.

Folgende Befehle führen gleichermaßen zum gewünschten Ergebnis:

```
. sum mpg if rep== 4 | rep == 5
. sum mpg if rep>= 4 & rep <= 5
. sum mpg if rep > 3 & rep ~= .
```

Slide 14

Die Gewichtsanweisung

Die Gewichtsanweisung führt dazu, dass jeder Beobachtung des Datensatzes ein bestimmtes Gewicht zugewiesen wird. Die Gewichtsanweisung hat folgende Syntax:

```
[gewichtungstyp = exp]
```

Dabei sind drei Gewichtungstypen zugelassen:

- *fweight* frequency weights
- *aweight* analytic weights
- *pweight* sampling weights

Wenn lediglich `weight` als Gewichtungstyp angegeben wird, so wird der für das jeweilige Kommando „übliche“ Gewichtungstyp verwendet. Manche Kommandos kennen darüber hinaus den Gewichtungstyp `iweight`, importance weights, der aber keine einheitliche statistische Bedeutung hat. Jedes Kommando, dass „importance weights“ zuläßt wird dies auf eine spezifische Weise tun. Hier muß man das Handbuch konsultieren. Die drei übrigen Gewichtungstypen haben eine klare statistische Bedeutung:

- *Frequency Weights* werden verwendet für Gewichtungsvariablen, welche angeben, wie oft eine Beobachtung im Datensatz vorkommt.
- *Analytic Weights* werden verwendet für Aggregatdaten, bei denen jeder Fall eine Maßzahl darstellt, welcher eine unterschiedliche Anzahl von Fällen zugrundeliegt.
- *Sampling Weights* ist für Daten aus Stichproben, bei denen die Elemente mit unterschiedlicher Wahrscheinlichkeit aus der Grundgesamtheit gezogen wurden.

Slide 15

Optionen

Nahezu jeder Stata–Befehl kann durch sogenannte „Optionen“ näher spezifiziert werden. Optionen werden durch ein Komma vom eigentlichen Kommando abgetrennt. Bei mehreren Optionen steht *nur ein* Komma. Das Komma leitet also nicht eine Option, sondern eine Liste von Optionen ein. Die Reihenfolge der Liste der Optionen eines Befehls ist beliebig.

▷ **Beispiel**

```
. tabulate rep for, mis row col
. summarize mpg, detail
. describe, short
```

Üblicherweise stehen Optionen am Ende eines Kommandos, also nach dem Befehl, der Variablenliste, den if/in–Bedingungen und der Gewichtsanweisung. Optionen können aber innerhalb eines Kommandos an jeder Stelle *nach* der Variablenliste stehen. Wenn Sie nach den Optionen weitere Befehlsteile spezifizieren wollen, müssen Sie die Optionen mit einem Komma beenden.

Die meisten Optionen lassen sich abkürzen. Über das Ausmaß der Abkürzungsmöglichkeit informiert die Online–Hilfe. Der Teil einer Option, der mindestens eingegeben werden muß, ist farblich hervorgehoben.

Slide 16

Das Befehlsprefix „by“

Der Prefix „by“ besteht aus dem Prefix und einer Variablenliste. Es bewirkt, daß der eigentliche Stata–Befehl für alle Kategorien der Variablen aus der Liste wiederholt wird.

Die Anwendung des „By“–Prefixes setzt voraus, daß der Datensatz nach den Variablen der „By“–Liste sortiert wurde.

▷ **Beispiel**

```
. sort for rep78
. by for: sum mpg
. by for rep78: sum mpg
```

Slide 17

Das Befehlsprefix „for“

Das Befehlsprefix `for` kann vor jeden Stata–Befehl gestellt werden. Es besteht aus dem Befehlsprefix selbst, der Definition des Listentyps, einer Liste von Parametern („forlist“) und verschiedenen Optionen. Es dient dazu, den eigentlichen Stata–Befehl nacheinander mit allen Parametern der „forlist“ durchzuführen. Dazu wird in den eigentlichen Stata–Befehl der Platzhalter „X“ eingebaut. Dieser Platzhalter wird sukzessiv durch die Parameter der „forlist“ ersetzt.

▷ **Beispiel**

```
. for varlist rep78 trunk hdroom: tabulate X foreign
```

Da `for` vor *jeden* Befehl gestellt werden kann, wird dieser Baustein in den Syntaxangaben der Befehle nicht aufgeführt.

Der Befehl des Beispiels führt nacheinander folgende Befehle aus:

```
. tabulate rep78 foreign
. tabulate trunk foreign
. tabulate hdroom foreign
```

Mit dem Wort `varlist` im obigen Befehl wird der *Listentyp* spezifiziert. Hier handelt es sich um eine Variablenliste, nämlich die Variablen *rep78, trunk* und *hdroom*.

▶ Beispiele für die Spezifiktation anderer Listentypen finden sich auf der nachfolgenden Folie.

Slide 18

Die „forlist“

Die Parameter der „forlist“ können sein: Eine Variablenliste, Zahlen oder eine beliebige Liste von Parametern. Um welche Art der „forlist“ es sich handelt wird *vor* der Parameterliste festgelegt. Wenn es sich um eine Liste von Variablen handelt, wird `varlist` bzw. `var` angegeben. Bei Zahlen wird `num` angegeben, bei neuen Variablen `new` und für Wort- oder Buchstabenlisten `any`.

▷ **Beispiel**

```
. for num 1/10: generate rX=uniform()
. for any a b c d e f: generate rX = uniform()
. ds
```

Der Befehl `generate` dient zur Bildung neuer Variablen. Im ersten Kommando werden darum 10 gleichverteilte Zufallsvariablen mit den Namen r1 r2 ... r10 erzeugt. Im zweiten Befehl werden entsprechende Variablen mit den Namen *ra*, *rb* usw. erzeugt.

Slide 19

Komplexe *for*–Befehle

„for“ erlaubt die Angabe von bis zu 9 „forlists“. Die unterschiedlichen Listen werden dabei durch einen *backslash* (\) voneinander abgetrennt. Die Elemente der ersten Liste werden durch `X`, die Elemente der zweiten Liste durch `Y` und die Elemente weiterer Listen durch `Z,A,B...F` angesprochen.

Darüberhinaus können mehrere Befehle hinter dasselbe for–Prefix gestellt werden. Diese werden ebenfalls durch einen *backslash* getrennt.

▷ **Beispiel**

```
. for var mpg weight length \ newlist verb gew laenge: sum X \
gen Y = X - r(mean)
```

Hiermit werden die neuen Variablen *verb*, *gew* und *laenge* erzeugt, welche die *zentrierten* Werte der Variablen *mpg*, *weight* und *length* enthalten.

In der ersten *Runde* werden folgende Befehle ausgeführt:

```
. sum mpg
. gen verb = mpg - r(mean)
```

Zunächst werden Kennzahlen der Verteilung der Variable *mpg* berechnet. Anschließend wird die neue Variable *verb* erzeugt. In der zweiten Runde wird *mpg* durch *weight* und *verbr* durch *gew* ersetzt.

Der Ausdruck *r(mean)* steht für eine *gespeichertes* Ergebnis. Jedes *Statistik–Kommando* von Stata speichert seine wichtigsten Ergebnisse als sogenannte *interne Ergebnisse*. Andere Befehle können diese internen Ergebnisse ansprechen. Oben wird das interne Ergebnis *r(mean)* angesprochen. Es steht für den Mittelwert der zuletzt verwendeten Variable von *summarize*. Von der Variable *mpg* wird somit der Mittelwert der Variable *mpg* abgezogen, d.h. die Variable wird *zentriert*.

Durch den Befehl

```
. return list
```

nach einem Statistikkommando erhalten Sie eine Liste der gespeicherten internen Ergebnisse des Kommandos.

▶ Wenn Sie häufiger mit komplexen *for*–Befehlen arbeiten, sollten Sie sich mit dem Befehle `while` beschäftigen. `while` erlaubt die Programmierung von Schleifen und ist sowohl schneller als auch übersichtlicher als komplexe *for*–Befehle.

Slide 20

Kommandos, die man kennen sollte

Zusätzlich zu den hier präsentierten Kommandos sollte man sich mit folgenden Befehlen vertraut machen:

```
. pwd           . codebook          . table
. cd            . browse
. copy          . count
. mkdir         . inspect
. type
. shell
. winexec
```

3 Do–Files

Slide 21

Do–Files

Alle Stata–Befehle können sowohl interaktiv eingegeben als auch in eine Befehlsdatei geschrieben werden. Die Befehlsdatei kann mit jedem beliebigen Editor geschrieben werden. Das Format der Datei muß *Plain ASCII* sein.

Wenn Sie unter Windows 95, 98 oder NT arbeiten können Sie mit Befehl

```
. doedit
```

den Stata Do–File Editor aufrufen.

Nach Aufruf des Stata Do–File Editors können beliebige Befehle in den Editor geschrieben werden. Man kann die eingegebenen Befehle direkt aus dem Editor starten (Menü: **tools**) oder die eingegebenen Befehle als Datei speichern und dann von der Kommandozeile aus starten.

Wird anstelle des Stata Do–File Editors ein anderer Editor verwendet ist darauf zu achten, dass der Do–File nur dann ausgeführt werden kann, wenn der Do–File zuvor gespeichert wurde. Es wird immer diejenige Version des Do–Files ausgeführt, die zuletzt gespeichert wurde.

Wird ein Editor verwendet, in dem mehrere Dateien gleichzeitig geöffnet sein können (z.B. Word, PFE, UlraEdit usw.) können alle diese Dateien von Stata angesprochen werden, wenn Sie zuvor gespeichert wurden.

Slide 22

Wie wird eine Befehlsdatei gestartet?

Befehlsdateien werden mit dem Befehl `do Dateiname` gestartet. Als Dateiname wird der Name der jeweiligen Befehlsdatei verwendet. Wird keine Erweiterung angegeben, wird die Erweiterung *do* vorausgesetzt.

▷ **Beispiel**

```
. do an1
```

startet die Befehlsdatei *an1.do*.

Der Befehl `do` kann auch innerhalb von Do–Files verwendet werden, d.h. Do–Files können ineinander verschachtelt werden.

Der Befehl im Beispiel kann nur ausgeführt werden, wenn Sie zuvor eine Datei mit dem Namen *an1.do* erstellt haben. Diese Datei können Sie mit dem Stata Do–File Editor oder mit jedem anderen Editor erstellen.

Als Dateiname kann jeweils auch die komplette Pfadangabe verwendet werden.

Es empfiehlt sich, die Ergebnisse eines *Do–Files* in einer Ergebnisdatei aufzuzeichnen. Diese wird durch `. log using Dateiname` geöffnet und durch `. log close` geschlossen.

Slide 23

Befehle, die in keinem Do–File fehlen sollten

Jeder Do–File sollte mit folgenden Befehlen beginnen:

```
version 6.0
set more off
capture log close
log using Dateiname, replace
```

Am Ende jedes Do–Files sollten folgende Befehle stehen:

```
log close
exit
```

Mit `version` wird angegeben, für welche Stata–Version der Do–File geschrieben wurden. Hierdurch wird sichergestellt, dass der Do–File auch unter späteren Versionen noch läuft.

Mit `set more off` wird die Bildschirmweise Ausgabe unterbrochen.

Durch `log close` werden geöffnete Ergebnisdateien geschlossen. Durch Voranstellen von `capture` vermeidet man ein Fehlermeldung, wenn keine Log–Datei geöffnet ist. Durch `log using Dateiname` wird eine Ergebnisdatei geöffnet. Die Option `replace` sorgt dafür, dass alte Ergebnisdateien des selben Namen überschrieben werden.

Mit `log close` am Ende des *Do–Files* wird die Ergebnisaufzeichnung wieder beendet. `exit` ist bedeutungslos. Es dient lediglich dazu, dass der letzte Befehl im *Do–File* mit einem *harten Return* abgeschlossen wird.

Slide 24

Interaktive Arbeit in *Do–Files* umwandeln

Interaktive eingegebene Befehle können einfach in *Do–Files* umgewandelt werden. Hierzu wird die Ausgabe des Befehls

```
. #review 100
```

in einer Ergebnisdatei aufgezeichnet und mit dem Editor bearbeitet.

Eine andere Möglichkeit ist die Verwendung der Option `noproc` des Befehls `log using`. Durch sie werden nur die Befehle, nicht aber die Ergebnisse in einer Ergebnisdatei aufgezeichnet.

Slide 25

Lange Zeilen in *Do–Files*

Befehle in *Do–Files* werden durch *Return* abgeschlossen. Um längere Befehle über mehrere Zeilen umzubrechen kann auf das Semikolon als Befehlsende umgeschaltet werden:

```
#delimit ;
for any a b c d e f:
    generate rX = uniform();
#delimit cr
```

Alternativ kann das Zeilenende *auskommentiert* werden:

```
for any a b c d e f: /*
*/ generate rX = uniform()
```

Slide 26

Die Datei *profile.do*

Ein spezieller *Do–File* ist *profile.do*. Die Befehle dieses *Do–Files* werden beim Aufruf von Stata automatisch ausgeführt. Hierdurch kann man die Eigenschaften von Stata den individuellen Bedürfnissen anpassen.

▷ **Beispiel**

Eine typische *profile.do*–Datei:

```
set matsize 100
global F4 "dir *.dta;"
global F5 "dir *.do;"
global F6 "codebook;"
global F9 "do _marked;"
```

Damit *profile.do* von Stata ausgeführt werden kann, muß sich die Datei im aktuellen Arbeitsverzeichnis oder in einem Verzeichnis dass von Stata durchsucht wird befinden. Ein guter Platz für *profile.do* ist `c:/ado/personal`.

Mit dem Befehl `set matsize 100` wird die maximale Größe von bearbeitbaren Matrizen auf 100×100 eingestellt. Die Voreinstellung ist 40×40.

Mit `global` werden sogenannte globale Makros erstellt. Diese werden vor allem innerhalb von Programmen benötigt. Der Befehl kann aber auch dazu genutzt werden, um die Funktionstasten mit beliebigen Stata–Befehlen zu belegen. `global F4 "dir*.dta";` belegt die Funktionstaste „F4“ mit dem Befehl `dir *.dta`. Dies bewirkt eine Auflistung der Stata–Systemdatensätze im Arbeitsverzeichnis.

Das Semicolon in den einzelnen Befehlen bewirkt, dass der Befehl unmittelbar durch die Funktionstaste ausgeführt wird. Ohne das Semicolon muß der Befehl noch mit der Eingabetaste abgeschlossen werden.

Die Funktionstaste „F9“ wird oben mit dem Befehl `do _marked` belegt. Dies macht dann Sinn, wenn der für *Do–Files* verwendete Editor so eingestellt wurde, dass markierte Bereiche mit der „F9“–Taste unter dem Namen *_marked.do* abgespeichert wird.

Slide 27

Kommandos, die man kennen sollte

Zusätzlich zu den hier präsentierten Kommandos sollte man sich mit folgenden Befehlen vertraut machen:

```
. run
. more
```

4 Einlesen von Rohdaten mit Stata

Slide 28

Überprüfung des Datenformats

Vor Einlesen von Rohdaten sollte das Datenformat überprüft werden. Dies geschieht am einfachsten mit `type`. Durch `type Dateiname` wird der Inhalt der Datei *Dateiname* am Bildschirm angezeigt.

▷ **Beispiel**

```
. type c:/programme/stata/auto.dta
. type http://jse.stat.ncsu.edu:70/00/jse/data//titanic.dat
```

Slide 29

Freies Format, keine Strings

Rohdaten im freien Format können ohne *Data–Dictionary* eingelesen werden. Diese Datensätze haben haben Leerzeichen zwischen den Variablen.

▷ **Beispiel**

```
. type http://jse.stat.ncsu.edu:70/00/jse/data//titanic.dat
1        1        1        1
1        1        1        1
1        1        1        1
```

Dies ist ein Datensatz im freien Format mit 4 Variablen. Er kann wie folgt eingelesen werden:

```
. infile v1 v2 v3 v4 using
http://jse.stat.ncsu.edu:70/00/jse/data//titanic.dat
```

(Denken Sie daran: Befehle können durch die *„Bild–nach–Oben "* Taste in das Kommandofenster zurückgeholt werden. Sie brauchen also nicht jedesmal die gesamte Adresse eingeben.)

Bei Datensätzen im freien Format ist es wichtig, die Zahl der Variablen zu kennen. Im Befehl `infile` werden dann die Namen der Variablen einzeln aufgezählt. Dabei können beliebige Namen verwendet werden.

Der Dateinamen wird hinter dem Kennwort `using` eingegeben.

Vor Verwendung von `infile` müssen etwa vorhandene Daten aus dem Arbeitsspeicher gelöscht werden. Dies erreicht man durch

```
. clear
```

oder

```
. drop _all
```

Slide 30

Freies Format, *Strings* in Anführungszeichen

▷ **Beispiel**

```
. type http://jse.stat.ncsu.edu:70/00/jse/data//baseball.dat

3300 0.272 0.302  69 153 21  4 31 104  22  80  4  3 1 0 0 0 "Andre Dawson     "
2600 0.269 0.335  58 111 17  2 18  66  39  69  0  3 1 1 0 0 "Steve Buchele    "
2500 0.249 0.337  54 115 15  1 17  73  63 116  6  5 1 0 0 0 "Kal Daniels      "

. infile v1-v17 str30 v18 using
http://jse.stat.ncsu.edu:70/00/jse/data//baseball.dat
```

Die 18. Variable des Datensatzes ist eine *String–Variable*. Die *Strings* sind in Anführungszeichen eingeschlossen. Derartige Variablen müssen in der Variablenliste des `infile`–Befehls durch das Kürzel `str` kenntlich gemacht werden. Die Zahl 30 gibt an, dass bis zu 30 Buchstaben auftreten. Maximal kann `str80` verwendet werden.

Wenn Variablen einfach durchnummeriert werden sollen, kann dies durch *von–bis* Variablenlisten (z.B. `v1-v17`) erreicht werden.

Slide 31

Freies Format, *Strings* ohne Leerzeichen

▷ **Beispiel**

```
. type http://jse.stat.ncsu.edu:70/00/jse/data//93cars.dat

Acura         Integra        Small   12.9 15.9 18.8 25 31 0 1 4 1.8 140 6300
2890 1 13.2 5 177 102 68 37 26.5 11 2705 0
Acura         Legend         Midsize 29.2 33.9 38.7 18 25 2 1 6 3.2 200 5500
2335 1 18.0 5 195 115 71 38 30.0 15 3560 0
```

```
. infile str20 manu str20 make str2 type v1-v23 using
http://jse.stat.ncsu.edu:70/00/jse/data//93cars.dat
```

Datensätze, bei denen *String–Variablen* nicht in Anführungszeichen eingeschlossen sind können wie Datensätze mit Anführungszeichen behandelt werden, *wenn keine Leerzeichen innerhalb der Strings auftreten*.

Treten Leerzeichen innerhalb der Strings auf, muß ein *Data Dictionary* verwendet werden.

▷ **Beispiel**

```
. type http://jse.stat.ncsu.edu:70/00/jse/data//airport.dat

HARTSFIELD INTL      ATLANTA               285693 288803 22665665 165668.76  93
> 039.48
BALTO/WASH INTL      BALTIMORE              73300  74048  4420425  18041.52  19
> 722.93
LOGAN INTL           BOSTON                114153 115524  9549585 127815.09  29
> 785.72
```

Der Befehl

```
. infile str60 name str30 town v1-v5 using
http://jse.stat.ncsu.edu:70/00/jse/data//airport.dat
```

führt zu Problemen, da das Leerzeichen zwischen *Hartsfield* und *Intl* als Beginn einer neuen Variablen interpretiert wird.

Slide 32

Spreadsheet Format

▷ **Beispiel**

```
. type http://jse.stat.ncsu.edu:70/00/jse/data//usnews.dat

1061,Alaska Pacific University,AK,2,490,482,972,20,440,530,430,550,18,22,193,14
> 6,55,16,44,249,869,7560,7560,4120,1620,2500,130,800,1500,76,72,11.9,2,10922,1
> 5
1063,University of Alaska at Fairbanks,AK,1,499,462,961,22,*,*,*,*,*,*,1852,142
> 7,928,*,*,3885,4519,1742,5226,3590,1800,1790,155,650,2304,67,*,10.0,8,11935,*

. insheet using
http://jse.stat.ncsu.edu:70/00/jse/data//usnews.dat
```

Beim *Spreadsheet Format* werden die Variablen durch Kommas abgetrennt. Für derartige Daten kann der Befehl `insheet` verwendet werden. Bei diesem Befehl kann die Variablenliste entfallen. Dann werden die Variablen einfach durchnummeriert.

Werden andere Variablennamen gewünscht, müssen diese angegeben werden.

Manchmal stehen im *Spreadsheet* Format zwischen den Variablen Tabulatoren statt Kommas. Diese Daten können ebenfalls mit `insheet` eingelesen werden. Allerdings sehen diese Daten wie Daten im freien Format aus. Mit der Option `showtabs` des `type`–Befehls werden Tabulatoren optisch sichtbar gemacht.

Slide 33

Fixed Format

▷ **Beispiel**

```
. type http://jse.stat.ncsu.edu:70/00/jse/data//airport.dat

HARTSFIELD INTL      ATLANTA              285693 288803 22665665 165668.76  93
> 039.48
BALTO/WASH INTL      BALTIMORE             73300  74048  4420425  18041.52  19
> 722.93
LOGAN INTL           BOSTON               114153 115524  9549585 127815.09  29
> 785.72
```

In diesem Datensatz gibt es für Stata keinen Anhaltspunkt dafür, wo eine Variable beginnt, und wo sie aufhört. Dies muß dem Programm mitgeteilt werden. Hierzu wird eine Steuerkartendatei (*Data–Dictionary*) benötigt.

Slide 34

Das *Data–Dictionary*

Die Steuerkarte zum Einlesen von Rohdaten im *Fixed Format* besteht aus dem Stata–Befehl `dictionary`. Der Befehl steht entweder in der Datei der Rohdaten selbst, oder in einer externen Datei. Letzteres ist empfehlenswert, da die Rohdaten dann auch von anderen Programmen lesbar bleiben.

In der Steuerkartendatei müssen folgende Angaben enthalten sein:

- Anzahl und Name aller Variablen
- Variablentyp jeder Variable
- Genaue Position der Variablen im Datensatz

Zusätzlich zu den notwendigen Angaben kann das Variablenlabel und der Name der Wertelabel spezifiziert werden. Näheres zu den Wertelabeln erfahren Sie im folgenden Abschnitt.

Wenn das *Data–Dictionary* in eine externe Datei geschrieben wird, so erhält diese Datei die Erweiterung *.dct*.

Slide 35

Syntax von `dictionary`

Am häufigsten wird der dictionary in folgender Form verwendet:

```
dictionary [using dateiname] {
    [Anfangsposition VAR1]
        [Variablentyp] Variablenname Variablenbreite] ["Variable Label"]
    [Anfangsposition VAR2]
        [Variablentyp] Variablenname Variablenbreite] ["Variable Label"]
    ...
}
```

Die Angaben in der Klammer können unter bestimmten Umständen weggelassen werden.

Die Steuerkarte beginnt mit dem Befehl `dictionary`. Mit `[using dateiname]` wird angegeben, welche Rohdatendatei gelesen werden soll. Diese Angabe wird weggelassen, wenn die Rohdaten dem Dictionary direkt nachfolgen.

Die Mengenklammer eröffnet die eigentliche Beschreibung der Daten. Danach wird zunächst die Anfangsposition der ersten Variablen eingegeben.

Die erste Variable der *Airport*–Daten beginnt in Spalte 1. Dies wird durch

`_column(1)`

spezifiziert. Die Angabe kann weggelassen werden, da Spalte 1 die Voreinstellung ist.

Anschließend erfolgt der Name des Variablentyps. In der Regel genügt es zwischen numerischen Variablen und *String*–Variablen zu unterscheiden. Bei *numerischen* Variablen kann der Variablentyp weggelassen werden. Bei *String*–Variablen muss **`str#`** eingegeben werden. # steht für die Anzahl der Spalten, die der Variable *eingeräumt* werden.

Die erste Variable der *Airport*–Daten ist eine *String–Variable*. Ihr sollen 12 Spalten eingeräumt werden:

`str12`

Nun folgt der Variablenname. Hier wird

```
airport
```

verwendet.

Als Variablenbreite wird die Anzahl der Spalten angegeben, welche die Variable einnimmt. %6 bedeutet 6 Spalten, %12 12 Spalten usw. Nach der Spaltenzahl steht ein „f“ bei numerischen Variablen und ein „s“bei *String*–Variablen. Die Angabe muß mit der Breite in der Rohdatenmatrix *exakt* übereinstimmen.

Die erste Variable der *Airport*–Daten nimmt 21 Spalten ein. Daher wird

```
%20s
```

eingegeben.

Als Variablenettikett kann ein Text mit 64 Zeichen Längen verwendet werden. Der Text muß mit Anführungszeichen eingeschlossen werden.

Die Anfangsposition der 2. Variable wird identisch spezifiziert wie bei Variable 1. Die Angabe kann jedoch weggelassen werden, wenn sie sich aus der Anfangsposition und der Variablenbreite der vorangegebenen Variable ergibt.

Bei den *Airport*–Daten kann

```
_column(22)
```

eingegeben werden oder diese Angabe weggelassen werden.

Slide 36

Dictonary der *Airport*–Daten

Die Datei *airport.dct*:

```
dictionary using http://jse.stat.ncsu.edu:70/00/jse/data//airport.dat {
             str12 airport %21s
             str12 city    %21s
 _column(44)       sdep    %7f "Scheduled Departures"
 _column(51)       pdep    %7f "Performed Departures"
 _column(58)       pass    %9f "Enplaned Passengers"
 _column(67)     freight   %9f "Enplaned tons of freight"
 _column(77)       mail    %9f "Enplaned tons of mail"
}
```

Nach Speichern des *Data–Dictionary* können die Daten durch

```
. infile using airport
```

geladen werden.

Wenn der Befehl `infile` ohne Variablenliste spezifiziert wird, heißt dies, dass ein *Data–Dictionary* gegeben wird. Hinter `using` wird dann die Datei angegeben, welche das *Data–Dictionary* enthält. Voreingestellte Dateinamenerweiterung ist *.dct*

Das vorgeführte *Dictionary* zeigt nur die wesentliche Grundstruktur. Weitere Möglichkeiten finden sich unter

```
. h infile2
```

Slide 37

Eingabe nicht–maschinenlesbarer Daten

Tabellen aus gedruckten Quellen können mit folgenden Techniken eingegeben werden:

- Durch den Stata–Dateneditor. Dieser wird durch `edit` aufgerufen.
- Durch den Befehl `input`.

▶ Für den nachfolgenden Abschnitt sollten Sie versuchen mit Hilfe der hier vorgestellten Werkzeuge, folgende Dateien in Stata zu laden:

`Genex1.dta`

	persnr	intnr
1.	78205	19
2.	10304	19
3.	528601	19
4.	78203	19
5.	10303	110
6.	10302	287
7.	166604	287
8.	150401	287

`Genex2.dta`

	hhnr	eink
1.	1	0
2.	1	1100
3.	1	2600
4.	2	0
5.	2	6000
6.	3	3300
7.	4	620
8.	4	1100

Slide 38

Kommandos, die man kennen sollte

Zusätzlich zu den hier präsentierten Kommandos sollte man sich mit folgenden Befehlen vertraut machen:

```
. infix          . notes
```

▶ Bitte laden Sie nun den *Auto*–Datensatz wieder in den Arbeitsspeicher:

```
. use c:/programme/stata/auto, clear
```

5 Variablen erzeugen und verändern

Slide 39

Die Befehle „generate“ und „replace“

Der Befehl `generate` dient dazu, Variablen neu zu erstellen, der Befehl `replace` verändert den Inhalt einer bereits vorhandenen Variable. Die Syntax der beiden Befehle ist *vollkommen* identisch:

```
[by varlist:]  generate newvar = exp [if] [in]
[by varlist:]  replace oldvar = exp [if] [in]
```

▷ **Beispiel**

```
. generate newvar = 1
. replace newvar = 0
```

Im Beispiel wird zunächst eine Variable erstellt, die bei allen Beobachtungen den Wert 1 hat. Danach wird die gerade neu erstellte Variable verändert. Anstatt den Wert1 hat die Variable nun für alle Fälle den Wert 0.

Slide 40

Variablennamen

Die Namen von Variablen, die mit `generate` gebildet werden können aus acht Zeichen bestehen. Als Zeichen dürfen Buchstaben (A-Z und a-z), Zahlen (0-9) und der Unterstrich (_) verwendet werden. Folgende Namen sind nicht erlaubt:

`_all`	`double`	`long`	`_rc`
`_b`	`float`	`_n`	`_se`
`byte`	`if`	`_N`	`_skip`
`_coef`	`in`	`_pi`	`using`
`_cons`	`int`	`_pred`	`with`

Einige Variablennamen sind erlaubt, aber nicht empfehlenswert. So sollte man den Variablennamen „e“ vermeiden, da er an einigen Stellen mit dem „e“ der wissenschaftlichen Schreibweise von Zahlen verwechselt werden kann. Weiterhin sollten man Namen vermeiden, die mit einem Unterstrich beginnen, da interne Stata–Variablen mit einem Unterstrich beginnen und es deshalb zu Konflikten kommen kann. Schließlich empfiehlt es sich, auf Namen die mit einem „I“ beginnen zu verzichten, da der Befehl „xi“ diese Variablen löscht.

Slide 41

Beispiele für `generate` und `replace`

```
. generate verbr = 100 * 3.78/(1.6*mpg)
. gen logprice = log(price)
. replace logprice = log(price)/log(2)
. generate r = uniform()
. generate r01 = invnorm(uniform())
```

Das erste Kommando rechnet den Verbrauch von *Meilen pro Gallone* in *l/(100 km)* um. Das zweite Kommando berechnet den *natürlichen* Logarithmus des Preisen, das dritte den Logarithmus zu Basis zwei. Danach wird zunächst eine gleichverteilte Zufallsvariable, anschließend eine standard–normalverteilte Zuvallsvariable berechnet.

In „generate“ bzw. „replace“ können alle unter `. h functions` aufgeführten *Funktionen* und alle unter `. h operators` aufgeführten Operatoren verwendet werden. Dies schließt auch die logischen Operatoren ein.

Slide 42

`generate` und logische Operatoren

In Stata haben falsche Ausdrücke den Wert 0, wahre den Wert 1. Aus diesem Grund ist z.B. folgender „generate“–Befehl zulässig:

```
. gen goodrep = rep78 == 5
```

`rep78 == 5` ist falsch, d.h. 0, für alle Fälle, die keinen *Repair–Record* von 5 haben und und wahr, d.h. 1, für alle Fälle, mit einem Repair–Record von 5. Das Kommando erzeugt darum die Variable *goodrep* mit dem Wert 1 für Autos mit gutem Repair–Record und 0 für alle anderen.

Mit

```
. tab goodrep
```

kann man sich diese neue Variable betrachten. Natürlich kann man dieselbe Vorgehensweise erweitern, z.B. durch

```
. gen batrep = rep78 == 1 | rep78 == 2
```

Wie bereits erwähnt ist Vorsicht geboten bei Befehlen wie diesem:

```
. replace goodrep = rep78 >= 4
```

Der Ausdruck ist auch für diejenigen Fälle war, für die *keine Angaben* über das Haushaltseinkommen vorliegen.

Slide 43

Variablenlabel

Variablenlabel werden mit dem Befehl `label variable` vergeben.

▷ **Beispiel**

```
. lab var rep78 "Pannenanfaelligkeit"
```

Der Befehl überschreibt das alte „Label“ der Variable durch „Pannenanfälligkeit“. Entsprechend würden man verfahren, wenn bisher noch kein „Label“ vergeben wäre.

Mit `. describe rep78` kann man sich das Ergebnis des Befehls betrachten.

Bei der Eingabe der Label kann man auf die Anführungszeichen verzichten, wenn das „Label“ keine besonderen Zeichen — Bindestriche, Leerzeichen, Kommata usw. — enthält. Umlaute und „ß“ werden nicht mit jedem verfügbaren Schriftsatz korrekt angezeigt. Man sollte sie darum nicht verwenden.

Die „Label“ können max. 80 Zeichen lang sein.

Weitere Informationen zu einer Variable können durch den Befehl `notes` abgelegt werden. Siehe dazu

```
. help notes
```

Slide 44

Value–Labels

Value–Labels werden in zwei Schritten vergeben:

1. Die Definition eines Behälters, der Werte und zugeordnete „Label“ enthält, durch `label define`
2. Die Verknüpfung dieses Behälters mit der gewünschten Variable durch `label value`

Die Schritte können in beliebiger Reihenfolge vorgenommen werden.

▷ **Beispiel**

```
. lab def highlow 1 "high" 5"low"
. lab val rep78 highlow
```

Im Beispiel wurde zunächst der Inhalt des Behälters „highlow“ definiert („label define“), danach wurde dieser Behälter mit der Variable „rep78“ verknüpft. Ein Behälter kann mit mehreren Variablen verknüpft werden.

Um Tabellen mit Label und Wert anzuzeigen kann man den Wert in das Label *einschließen*:

```
. lab def highlow 1 "high 1" 5"low 5", modify
```

Außerdem kann man mit dem Befehl `label list` den Inhalt eines oder mehrerer Behälter betrachten.

```
. label list highlow foreign
```

Die Eingabe von „label list“ ohne Angabe eines Behälternahmens führt zur Auflistung der Inhalte *aller* definierten Behälter.

Slide 45

`generate/replace` und if

Häufig wird `generate` und `replace` zusammen mit einer if–Bedingung verwendet.

▷ **Beispiel**

```
. gen weight_4 = 0
. replace weight_4 = 1 if weight > 0
. replace weight_4 = 2 if weight > 2200
. replace weight_4 = 3 if weight > 3190
. replace weight_4 = 4 if weight > 3600
```

Natürlich können die „if“–Bedingungen komplizierter gestaltet werden. Dasselbe gilt für die Werte, die den Variablen zugewiesen werden. Die Idee bleibt stets dieselbe. Die meisten Rekodierungsprobleme können durch solche einfachen Standardbefehle gelöst werden.

▶ Im folgenden werden spezielle Rekodierungskonzepte beschrieben. Umsteiger werden diese Konzepte aus anderen Programmen nicht kennen und deshalb etwas schwieriger zu lernen finden. Sie werden deshalb etwas ausführlicher dargestellt. Die behandelten Probleme sind recht fortgeschritten. Sie können diesen Teil überspringen und in Abschnitt 5.3 on page 60 weiterlesen.

Um die nachfolgenden Beispiele nachzuvollziehen muß ein anderer Datensatz verwendet werden. Bitte legen Sie Ihren aktuellen Datensatz zur Seite

```
. preserve
```

und laden Sie den Datensatz *genex1.dta* aus dem vorangegangenen Abschnitt:

```
. use genex1, clear
```

5.1 Recodieren mit „by:“, „_n“ und „_N“

Slide 46

Ein einführendes Beispiel

Es soll eine Variable erzeugt werden, die für jeden Interviewer angibt, wieviel Personen er bzw. sie befragt hat. Der Datensatz sieht so aus:

	persnr	intnr
1.	78205	19
2.	10304	19
3.	528601	19
4.	78203	19
5.	10303	110
6.	10302	287
7.	166604	287
8.	150401	287

Für jeden Befragten findet sich eine Angabe darüber, wer ihn interviewt hat. Man kann nun einfach feststellen, dass Interviewer Nr. 19 vier Interviews geführt hat, Interviewer Nr. 110 eines und Interviewer 287 drei.

Die gesuchte Variable sollte deshalb so aussehen:

	persnr	intnr	intcount
1.	78205	19	4
2.	10304	19	4
3.	528601	19	4
4.	78203	19	4
5.	10303	110	1
6.	10302	287	3
7.	166604	287	3
8.	150401	287	3

▶ Wie würden Sie diese Variable erzeugen? Denken Sie einen Augenblick darüber nach bevor Sie weiterlesen.

Slide 47

Lösung des einführenden Beispiels

```
. sort intnr
. quietly by intnr: generate intcount = _N
```

Die meisten Teile der Lösung sind bekannt. Unbekannt ist `quietly`, aber das ist unwichtig. `quietly` kann vor jedes Kommando gesetzt werden und dient dazu, die Ergebnisausgabe zu unterdrücken. Bleibt also `_N`.

- Um zu verstehen was `_N` ist, muß man verstehen was `_n` ist.

Slide 48

texttt_n

_n ist ein Platzhalter für die aktuelle Position einer Beobachtung im Datensatz. Mann kann _n nutzen, um den Datensatz mit einer laufenden Nummer zu versehen:

▷ **Beispiel**

```
. generate lfd = _n
```

	persnr	intnr	lfd
1.	78205	19	1
2.	10304	19	2
3.	528601	19	3
4.	78203	19	4
5.	10303	110	5
6.	10302	287	6
7.	166604	287	7
8.	150401	287	8

_n

Slide 49

_n und das Prefix by:

Zusammen mit dem Prefix `by:` ist `_n` die Position eines Falles innerhalb der jeweiligen Kategorie der „by–Variablen“.

▷ **Beispiel**

```
. quietly by intnr: gen intlfd =_n
```

	persnr	intnr	lfd	intlfd
1.	78205	19	1	1
2.	10304	19	2	2
3.	528601	19	3	3
4.	78203	19	4	4
5.	10303	110	5	1
6.	10302	287	6	1
7.	166604	287	7	2
8.	150401	287	8	3

Slide 50

_N und das Prefix by:

_N ist der höchste Wert von _n. Über alle Fälle betrachtet ist _N im Beispiel die Zahl 8.

Hinter by: steht _N für den höchsten Wert von _n innerhalb der jeweiligen Kategorie der „by–Variablen“. Im Beispiel hat „_N“ für den Interviewer Nr. 19 Wert „4“, für die Interviewer Nr. 110 den Wert von 1 usw. Der Befehl

```
. by intnr: generate intcount = _N
```

lautet damit für den ersten Interviewer:

```
. generate intcount = 4
```

für den zweiten Interviewer

```
. generate intcount = 1
```

usw.

Dies führt dann zum gewünschten Ergebnis:

	persnr	intnr	lfd	intlfd	intcount
1.	78205	19	1	1	4
2.	10304	19	2	2	4
3.	528601	19	3	3	4
4.	78203	19	4	4	4
5.	10303	110	5	1	1
6.	10302	287	6	1	3
7.	166604	287	7	2	3
8.	150401	287	8	3	3

Bevor Sie weiterlesen sollten Sie ihren ursprünglichen Datensatz mit

```
. restore
```

wiederherstellen.

► Recodierungen mit by:, _n und _N sind gewöhnungsbedürftig. Wenn man sich jedoch das Prinzip klar gemacht hat, ergeben sich schnell viele Anwendungsbeispiele. Hier nur ein weiteres (fiktives) Beispiel:

Gewünscht ist eine Variable für alle tatsächlich auftretenden Kombinationen zwischen Familienstand (*fam*) und Bildung (*bil*). Die Variable soll den Wert 1

für verheiratete Befragte („fam == 1“) mit Hauptschulabschluss („bil==1“) haben, den Wert 2 für verheiratete Befragte mit Realschulabschluss („bil == 2“) usw. Der „normale“ Weg dies zu tun ist

```
. generate fambil = 1 if fam == 1 & bil == 1
. replace fambil = 2 if fam == 1 & bil == 2
. <usw>
```

Je nach Menge der Codes müssen auf diese Weise eine große Anzahl von Kommandos eingeben werden. Mit den neu erlernten Konzepten reichen dagegen stats vier Zeilen:

```
. sort fam bil
. quietly by fam bil: replace fambil = 1 if _n == 1
. replace fambil = sum(fambil)
```

Warum? Stellen Sie sich folgenden (fiktiven) Datensatz vor:

```
          fam      bil   | schritt1   schritt2
  1.        1        1   |        1          1
  2.        1        1   |        .          1
  3.        1        1   |        .          1
  4.        1        1   |        .          1
  5.        1        1   |        .          1
  6.        1        2   |        1          2
  7.        2        1   |        1          3
  8.        2        1   |        .          3
  9.        2        1   |        .          3
 10.        2        2   |        1          4
 11.        2        2   |        .          4
 12.        2        2   |        .          4
```

Der Datensatz enthält die Variablen Familienstand und Bildung, wobei für jede der Variablen lediglich 2 Ausprägungen vorkommen. Der Datensatz ist nach Familienstand und Bildung sortiert. Mit dem Kommando „by fam bil: generate fambil = 1 if _n == 1“ wird die Variable in Schritt 1 erzeugt. Das Kommando weist dem ersten Fall („if _n == 1“) jeder Kombination der beiden „by–Variablen“ („by fam bil:“) den Wert 1 („generate fambil = 1“) zu. Das zweite Kommando berechnet $\sum_{i=1}^{n} \text{fambil}_i$. Berechnet wird also die laufende Summe der in Schritt 1 gebildeten Variable. Fehlende Werte werden dabei wie „0“ behandelt wird.

5.2 Rekodieren mit expliziten Subscripten

Slide 51

Explizite Subscripte

In Stata können Variablen mit Subscripten versehen werden. Subscripte werden in eckige Klammern eingeschlossen und direkt an den Variablennamen angehängt.

▷ **Beispiel**

```
. sort mpg
. display mpg[37]
. display mpg[_N/2 + 1]
. display 0.5 * (mpg[_N/2] + mpg[_N/2 + 1])
```

Der Befehl display zeigt das Ergebnis beliebiger Funktionen an. Das erste Kommando zeigt darum den Kraftstoffverbrauch des 37. Fahrzeuges im — nach dem Verbrauch sortierten — Datensatz an. Das zweite Kommando zeigt den Verbrauch des `_N/2 + 1`ten Fahrzeugs an, i.e. des $74/2+1 = 38$ten Fahrzeugs. Das letzte Kommando zeigt den mittleren Wert der beiden vorangegangenen Fahrzeuge i.e. der Median.

▶ Subscripte sind ein wichtiges Hilfsmittel zur Recodierung von Variablen. Sie werden vor allem dann eingesetzt, wenn die Angaben einer Beobachtung in eine oder mehrere andere Beobachtungen kopiert werden soll. Um das folgende Beispiel nachvollziehen zu können sollten sollten Sie ihren aktuellen Datensatz wieder mit

```
. preserve
```

zur Seite legen und den Datensatz *genex2.dta* aus dem vorangegangenen Abschnitt laden:

```
. use genex2, clear
```

Slide 52

Einführendes Beispiel

Gegeben sind Daten, in denen jeweils alle in einem Haushalt lebenden Personen interviewt wurden. Der Datensatz enthält Angaben zum Bruttoeinkommen.

	hhnr	eink
1.	1	0
2.	1	1100
3.	1	2600
4.	2	0
5.	2	6000
6.	3	3300
7.	4	620
8.	4	1100

Gewünscht wird nun eine Variable, die das Haushaltsbruttoeinkommen jedes Haushalts enthält.

Bitte denken Sie wieder ein wenig über die Lösung nach, bevor Sie weiterlesen.

Die Lösung lautet:

Slide 53

Lösung des einführenden Beispiels

```
. sort hhnr
. quietly by hhnr: generate hheink=sum(eink)
. quietly by hhnr: replace hheink = hheink[_N]
```

Das Kommando `generate hheink = sum(eink)` berechnet $\sum_{i=1}^{n} \text{eink}_i$. Aufgrund von `by:` wird jedoch nicht vom ersten bis zum letzten Fall des Datensatzes aufsummiert, sondern — „by hhnr:“ — vom ersten bis zum letzten Fall jedes Haushalts. In der letzten Beobachtung („_N“) jedes Haushalts steht danach die Summe der Bruttoeinkommen aller Haushaltsmitglieder. Diese Zahl wird durch „by hhnr: replace hheink = hheink[_N]“ auch den übrigen Personen im jeweiligen Haushalt zugewiesen.

Ein ähnlicher Fall ist die Zuweisung des Berufs des Haushaltsvorstands zu den übrigen Mitgliedern desselben Haushalts.

▶ Bitte stellen Sie jetzt Ihren ursprünglichen Datensatz mit

```
. restore
```

wieder her.

5.3 Spezielle Recodierungs–Befehle

Slide 54

recode

Die vielleicht häufigste Aufgabe bei Recodierungen ist das Zusammenfassen mehrerer Werte einer Variablen. Hierzu kann `replace` zusammen mit *if–Bedingungen* verwendet werden. Eine Abkürzung ist allerdings der Befehl `recode`

▷ **Beispiel**

```
. generate weight4 = weight
. recode weight4 min/2239 =1 2240/3189 =2 3190/3599 = 3
3600/max =4
```

Durch `recode` werden bestimmten Werten einer Variable gemäß einer Zuordnungsregel neue Werte zugewiesen. Oben wird z.B. den Werten „2240“ bis „3189“ („2240/3189“) der neue Wert „2“ zugewiesen. Statt eines Wertes können auch die Begriffe „min“ und „max“ verwendet werden, wobei „min“ den kleinsten und „max“ den größten Wert einer Variablen darstellt. Beachten Sie: „max“ steht *nicht* für den *fehlenden Wert* sondern stets für den höchsten inhaltlichen Wert!

Gegenüber Recodierungen mit „generate“ und „replace“ hat „recode“ den Nachteil, viel langsamer zu sein.

Eine andere Möglichkeit zum einfachen Zusammenfassen von Werten einer Variablen sind die *Funktionen* `recode()` und `autocode()`. Siehe hierzu `. h functions`

Um Variablen entlang von *Quantilen* zu kategorisieren eignen sich die Befehle `pctile` und `xtile`.

Slide 55

Der Befehl egen

Im Befehl „egen“ wird eine große — und ständig wachsende — Zahl von Erweiterungen des „generate“–Befehls zusammengefaßt. In ihrem Kern sind diese Erweiterungen nichts anderes als ein (oder mehrere) „generate“- und „replace“–Kommandos. Der Befehl „egen“ dient dazu, den Benutzer vom Nachdenken über die entsprechenden Kommandos zu entlasten.

Die Struktur von „egen“ gleicht derjenigen von „generate“. Nach dem Befehl folgt der Name der Variable, die *erzeugt* werden soll, dann ein Gleichheitszeichen und schließlich die sogenannte „egen–Funktion“. „egen–Funktionen“ können nur innerhalb des Befehls „egen“ angewandt werden.

Slide 56

Nützliche egen – Beispiele

▷ **Beispiel**

```
. egen repfor = group(rep78 foreign)
```

Bildet die Variable *repfor*, welche für jede Kombination der Variablen *rep78* und *foreign* einen anderen Wert enthält.

```
. egen pricem = mean(price), by(rep78)
```

Bildet *pricem*, welche den durchschnittlichen Preis für Autos mit unterschiedlichem *Repair Record* enthält.

```
. egen big = rsum(weight length hdroom displ)
```

Bildet die *fallweise* Summe der angegebenen Variablen.

```
. egen mis = rmis(weight length hdroom displ)
```

Bildet die *fallweise* Anzahl fehlender Werte.

Slide 57

Missing values

Fehlende Werte werden durch einen *Punkt* angesprochen und vergeben:

▷ **Beispiel**

```
. replace rep78=9 if rep78==.
. replace rep78=. if rep78==9
```

Um für mehreren Variablen gleichzeitig einen fehlenden Wert zu definieren kann das Prefix `for` oder der Befehl `mvdecode` verwendet werden. `mvencode` wird für die umgekehrte Richtung verwendet.

In Stata gibt es keinen Unterscheidung zwischen systemdefinierten und benutzerdefinierten fehlenden Werten wie etwa in SPSS. Zum vorübergehenden Ausschluß von Fällen mit bestimmten Werten können neue gebildete Variablen mit entsprechenden fehlenden Werten oder *if*–Bedingungen verwendet werden.

Slide 58

Kommandos, die man kennen sollte

Zusätzlich zu den hier präsentierten Kommandos sollte man sich mit folgenden Befehlen vertraut machen:

- `. rename`
- `. drop`
- `. keep`
- `. order`
- `. format`
- `. edit`
- `. encode`
- `. decode`
- `. separate`

Index